Estimating the Impact of Weather on CBOT Corn Futures Prices

Sriramjee Singh

Iowa State University

August 8, 2020

Job Market Paper

Abstract

We apply machine learning methods to weather and soil data to evaluate their impact on Chicago Board of Trade (CBOT) corn futures prices. We try to find out which weather information: historical, current or future affects the prices most. The weather information category which predicts the futures prices most precisely is considered as the most relevant information for the traders. We extend our study to the discovery of directional price movements in the market. The input variables, which include county-level, hourly weather data and county-level, time-independent soil data for all major corn producing US states, have spatial as well as temporal dimensions. We base our study on the reasoning that weather conditions affect corn production and expected production levels impact futures prices in the market. We use a coefficient-of-correlation test for features selection and employ a fully connected neural network on multiple inputs to forecast percentage change in daily futures prices. Our findings suggest that more specialized and focused training of the data by incorporating spatial attributes using a convolutional neural network delivers better results. Our study also shows that not all counties provide useful model information - some only create noise. Finally, we employ various classification models to find

future direction of CBOT prices, for which support vector machine and decision tree ensemble XGBoost show encouraging results.

1 Introduction

Weather is one of the most important factors of corn production in the United States. Favorable weather during the crop growing season raises production levels and, similarly, bad weather lowers production levels. The prevailing weather conditions during the season create expectations about yearly production levels - expectations that influence futures prices in the market. In short, weather conditions affect corn futures prices.

In 2019, heavy rains deluged Corn Belt farmers throughout April and May, which coincides with corn planting season. In a June 4, 2019 article, the Washington Post noted that the prospect of delayed planting pushed corn prices up by 20% since a mid-May low, and that prices could rise even more if the fear of a bad crop spread (Dam et al. [2019]). Similarly, higher production predictions can lower futures prices. An August 12, 2019 Bloomberg article reported that corn futures posted their worst rout since 2013 when the USDA pegged planted acres at 90 million, 2.6% higher than the average analysts estimate (Almeida et al. [2019]).

Several crop-yield prediction studies, especially for US corn and soybeans, use weather and soil data, and the vastness and complexity of weather data makes machine learning applications appropriate for these types of analyses. Jiang et al. [2018] forecast county-level corn yields using LSTM (long short-term memory), a special recurrent neural network (RNN) method, and provide a high-frequency (i.e., daily) yield expectation at a very granular (i.e., county) level. The results suggest that their prediction is more accurate than USDA's pre-harvest prediction. Khaki et al. [2019] find that a deep neural framework of CNN and RNN outperform all other methods like random forest and least absolute shrinkage and selection operator (LASSO) in forecasting corn and soybean yield across the US Corn Belt.

Local weather also affects investor behavior. Limpaphayom et al. [2007] find there is a correlation between Chicago weather and the behavior of S&P 500 index futures traders - the effective bid-ask spread is higher on windy days. Lu and Chou [2012] study the association between weather-related mood factors and stock index returns in China's Shanghai Stock Exchange, an order-driven market. Their results indicate that weather-induced mood changes do not affect asset returns; however, they find weather-related variables do strongly correlate with market turnover and liquidity. Overall, they show that in an order-driven market, environmental impacts on sentiment are likely to affect trading activities but not returns.

This article presents a unique opportunity for predicting Chicago Board of Trade (CBOT) corn futures prices as well as forecasting the price movement direction using environmental data, which could allow traders to rely less on anticipated production and instead estimate prices by exploiting readily available weather data. The direct linkage of weather and prices reduces the turnaround time of the model, which helps in such a highly fluid market.

The remainder of this paper is organized as follows. Section 2 summarizes related works of methodologies and application of machine learning and deep learning, and it presents existing works on futures prices prediction. Section 3 discusses the data used in the model training. Section 4 specifies details of our proposed machine learning models and data preprocessing methodology and illustrates forecasting models and classification models. Section 5 describes the training methodologies and experimental results. Finally, section 6 concludes and presents brief comments.

2 Literature Review

Crop yield prediction employing weather data is widespread in agricultural economics. High-frequency weather data invites machine learning techniques to interpret the entangled information. Kantanantha et al. [2010] use a weather-based regression model with time-dependent varying coefficients to predict annual yield in Hancock County Illinois and

develop a futures-based model for long-range cash price prediction. In their model, they predict cash price as a sum of the nearby settlement futures price and the projected commodity basis.

Price prediction has been a challenging task for researchers because of the noisy, complex, and non-stationary characteristics of data. Deep learning methods perform feature learning exercises more effectively in purposely designed networks. Long et al. [2018] propose a novel end-to-end model, the multi-filters neural network, specifically for feature extraction of financial time-series samples and price-movement prediction tasks. They integrate both convolutional and recurrent neurons to build the multi-filters structure to obtain information from different feature spaces and market views. Experimental results show that their network outperforms traditional machine learning and statistical models.

The generalized autoregressive conditional heteroskedasticity (GARCH) model is one of the most well-known price volatility forecasting models. To reduce the prediction error, Kristjanpoller and Minutolo [2015] apply the hybrid ANN–GARCH model in forecasting gold price volatility (spot and futures). The results show an overall improvement in forecasting using the hybrid when compared to GARCH alone. Kulkarni and Haidar [2009] build an optimal artificial neural network model through several sets of training based on lagged values of pre-processed spot and futures prices to forecast short-term crude oil spot-price direction. However, preselecting model inputs using feature selection algorithms before model training is crucial. Machine learning models without variable preselection are often inferior to time-series models when forecasting spot prices, which feature selection algorithms could reverse (Ceperic et al. [2017]).

Kim and Kim [2019] propose a model, composed of LSTM and CNN, which extracts temporal and image features from stock time-series and chart images, respectively, to predict stock prices. The results confirm that a combination of temporal and image features from the same data reduces prediction error when compared to using these features separately. Goutham et al. [2018] analyze heart rate variability signals obtained from electrocardiograms using a CNN and CNN-LSTM network to diagnose diabetes. The classification ac-

curacy obtained is significantly higher than previous studies in the field.

3 Data

The United States mostly produces corn in the Midwest region, which encompasses almost the entire Corn Belt. Our model uses hourly weather data, soil quality data, and soil moisture data from the US Midwest states - Iowa, Illinois, Indiana, Minnesota, Nebraska, Kansas, Michigan, Ohio, Missouri, and South Dakota - from 1980 to 2018.

Hourly weather data is from Weather Underground, a professional weather data company. Each data point represents 19x19 square miles area, which is more precise than commonly used weather station data. We choose raw data related to weather variables like precipitation, wind, and temperature. Precipitation is an important factor in corn plant development - enough rainfall in the growing season assures good yield, but floods or drought damage prospects.

Similarly, maximum, minimum, and average daily temperature strongly influence yield. Research confirms that temperatures between 50°F and 86°F (10°C and 30°C) are best for crop growth. The most common temperature index used to estimate plant development is growing degree days (GDD), which is calculated from the daily maximum and minimum air temperature. GDD has proven useful for crop consultants, producers, and scientists who uses it to predict plant development rate and growth stage. North Dakota Agricultural Weather Network Center calculates GDD as:

$$GDD = \frac{T_{max} + T_{min}}{2} - T_{base}$$

where, T_{max} = max[min(86°F, daily maximum temperature), 50°F], T_{min} = max(50°F, daily minimum temperature), and T_{base} is the base or threshold temperature required to trigger optimum growth (50°F for corn).

Soil moisture is a crucial factor affecting corn production as well. The Palmer Drought Severity Index (PDSI) is a long-term cumulative measure of water availability in the soil

that spans from -10 (dry) to +10 (wet), with zero being normal moisture conditions. PDSI uses temperature data and a physical water balance model to capture the basic effect of global warming on drought. The National Oceanic and Atmospheric Administration measures PDSI monthly at the crop reporting district level, and we match and assign PDSI values at the county level. Soil quality data is from Dr. Hendricks at Kansas State University. We collect data from the gSSURGO database (a database for storing gridded soil survey results) and aggregate it to the county level using only areas classified as cropland according to the National Land Cover Database. Thus, our data covers the whole Corn Belt at the county level and includes over 100 variables, each of which is time-invariant for each county, as we do not consider soil quality to change much over time. We pick 14 variables from the data that we think most closely relate to corn yield.

We collect historical corn yield data from the National Statistics Service (NASS) quick stats from 1980 to 2018. NASS collects corn yield data annually at the county level, which reveals years in which a county produces corn and years it does not. CBOT daily corn futures prices data are from Barchart. The data records daily open, close, high, and low prices. The futures trading hours at CME Globex start at 7:00 p.m. CST (Central Standard Time) on one day and close at 1:20 p.m. CST the next day. A 45-minute break in trading occurs from 7:45 a.m. to 8:30 a.m. CST. Barchart merges the prices of regular pit (opens in the morning at 8:30 a.m. CST) with electronic trading (opens at 7:00 p.m. CST, a day before). We analyze data from April to October for every year from 1980 to 2018, as this is the corn growing season in the Corn Belt.

4 Methodology

Our study includes two sets of mechanisms for a prediction and classification problem. We attempt various algorithms to find the best possible outcome. Below, we delineate the models we utilize.

4.1 Prediction Models

This paper uses Convolutional Neural Network (CNN) to predict futures prices. The historical weather data has spatial features as well in form of counties. The most challenging part is to fit the data into the CNN format. Figure 1 shows a regular CNN (image is taken from internet).

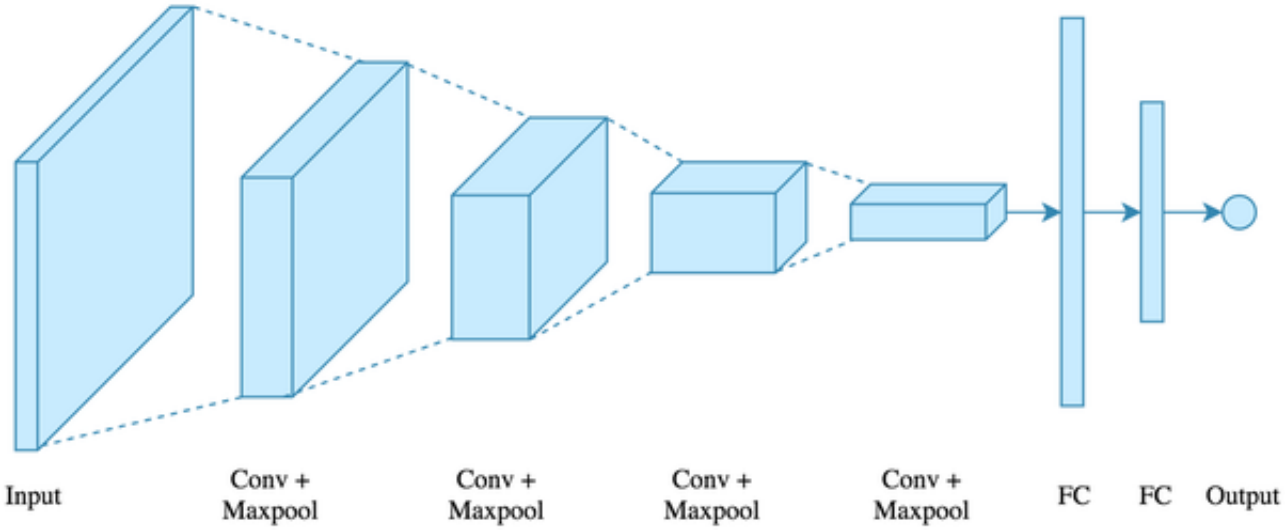

Figure 1: Convolutional Neural Network (CNN) Representation

CNNs are designed to automatically and adaptively learn spatial hierarchies of features through backpropagation by using multiple building blocks, such as convolution layers, pooling layers, and fully connected layers (Yamashita et al. [2018]). Convolution and pooling layers perform feature extraction, whereas a fully connected layer maps the extracted features into a final output, such as classification or forecasting. A convolution layer plays a key role in CNN, which is composed of a stack of mathematical operations, such as convolution, a specialized type of linear operation. As one layer feeds its output into the next layer, extracted features can hierarchically and progressively become more complex. Training is the process of optimizing parameters, such as kernels, to minimize the difference between outputs and true values, which is achieved through optimization algorithms like backpropagation and gradient descent, among others. A kernel is a small grid of parame-

ters and is an optimizable feature extractor.

A typical CNN architecture consists of repetitions of a stack of several convolution layers and a pooling layer followed by one or more fully connected layers.

Convolution layers are a fundamental component of the CNN architecture and perform feature extraction, which typically consists of a combination of linear and non-linear operations (i.e., convolution operation and activation function). The convolution is a dot product of the input, I and the convolution kernel, K. The output is a convolved feature map, f_c (Teow [2017]):

$$f_c = conv(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{1}$$

where, $*$ denotes a two-dimensional discrete convolution operator. Equation 1 shows that the convolution kernel, K, spatially slides over the input, tensor, I, to compute the element-wise multiplication and sum to produce an output, a convolved feature map, f_c.

The convolution operation we describe does not allow the center of each kernel to overlap the outermost element of the input tensor and reduces the height and width of the output feature map when compared to the input tensor. Padding, typically zero padding, is a technique to address this issue, where rows and columns of zeros are added on each side of the input tensor, so as to fit the center of a kernel on the outermost element and keep the same in-plane dimension through the convolution operation. The distance between two successive kernel positions, called a stride, defines the convolution operation. The common choice of a stride is 1; however, a stride larger than 1 is sometimes used in order to achieve downsampling of the feature maps. Pooling operations, described below, are an alternative technique for performing downsampling.

Kernel weight sharing is the only parameter automatically learned during the training process in the convolution layer, whereas the size of the kernels, number of kernels, padding, and stride are hyperparameters that need to be set before the training process starts. Outputs of the convolution operation the pass through an activation function, which can be a

linear function or non-linear function, such as sigmoid, hyperbolic tangent (tanh), or rectified linear unit (ReLU) function.

A pooling layer provides a typical downsampling operation, which reduces the in-plane dimensionality of the feature maps in order to introduce a translation invariance to small shifts and distortions and decrease the number of subsequent learnable parameters. Notably, there is no learnable parameter in any of the pooling layers, whereas filter size, stride, and padding are hyperparameters in pooling operations, similar to convolution operations. The most popular form of pooling operation is max pooling, which extracts patches from the input feature maps, outputs the maximum value in each patch, and discards all the other values.

The output feature maps of the final convolution or pooling layer are typically flattened (i.e., transformed into a one-dimensional (1D) array of numbers, or vector), and connected to one or more fully connected layers, also known as dense layers. In a fully connected layer, a learnable weight connects every input to every output. Once the features are extracted by the convolution layers and downsampled by the pooling layers, they are mapped by a subset of fully connected layers to the final outputs of the CNN network. The final fully connected layer typically has the same number of output nodes as the number of classes or number of prediction values. Each fully connected layer is followed by an activation function, as described above.

Training a network is the process of finding kernels in convolution layers and weights in fully connected layers, which minimize differences between outputs and actual values on a training dataset. A backpropagation algorithm is the method commonly used for training neural networks where loss function and gradient descent optimization algorithms play essential roles. A loss function through forward propagation on a training dataset calculates a model's performance under particular kernels and weights; and, learnable parameters, namely kernels and weights, are updated according to the loss value through backpropagation and gradient descent, among others.

A loss function, also referred to as a cost function, measures the disparity between outputs and true values of the dataset. Gradient descent is commonly used as an optimization algorithm that iteratively updates the learnable parameters (i.e., kernels and weights) of the network to minimize the loss. The gradient is, mathematically, a partial derivative of the loss with respect to each learnable parameter. We calculate a single update of a parameter as follows:

$$w := w - \alpha * \frac{\partial L}{\partial w} \tag{2}$$

where, w stands for each learnable parameter; α signifies a learning rate; and, L denotes a loss function. Notably, in practice, a learning rate is one of the most important hyperparameters to be set before the training starts. In addition, many improvements on the gradient descent algorithm have been proposed and widely used, such as SGD with momentum, RMSprop, and Adam.

Overfitting refers to a model learning statistical regularities specific to the training set. In other words, the model ends up memorizing the irrelevant noise instead of learning the signal, and, therefore, performs poorer on a subsequent new dataset. If the model performs well on the training set compared to the validation set, then the model is likely overfit to the training data. The best solution for reducing overfitting is to obtain more training data. The other solutions include regularization with dropout or weight decay, batch normalization, and data augmentation, as well as reducing architectural complexity.

Dropout is a regularization technique where randomly selected activations are set to 0 during the training so that the model becomes less sensitive to specific weights in the network. Weight decay, also referred to as $L2$ regularization, reduces overfitting by penalizing the model's weights so that the weights take only small values. Batch normalization is a type of supplemental layer that adaptively normalizes the input values of the following layer, mitigating the risk of overfitting, as well as improving gradient flow through the network, allowing higher learning rates, and reducing the dependence on initialization.

4.2 Classification Models

We attempted many classification algorithms before finally selecting two reassuring models.

4.2.1 Support Vector Machine (SVM)

Support vector machines (SVMs) are a powerful machine learning method and are, widely used in classification problems. We outline the SVM algorithm for binary classification, as required in our case (Lodha et al. [2006]). Binary classification uses a set of training data to create a classifier that correctly predicts the label of future examples that are not present in the training set. Let $S = (x_1, y_1), (x_2, y_2), ..., (x_m, y_m) \subseteq XxY$ be the m-example set where each x_i is an n dimensional feature vector and y_i is an integer class label, $-1, 1$. We suggest Scholkopf and Smola [2001] and; Burges [1998] for details that help readers understand our model.

Figure 2 shows a regular SVM (image is taken from internet). The goal is to first find a hyperplane (i.e., decision boundary) to separate the classes. Two parallel hyperplanes, also known as support vectors, are drawn on each side of the hyperplane separating the classes. Finally, the margin (i.e., the shortest distance) between the two extreme hyperplanes is maximized to ensure that classes do not overlap.
The equation for separating a hyperplane and the two support vectors is:

$$\begin{aligned} wx_i + b &= 0 \\ wx_i + b &= -1 \\ wx_i + b &= +1 \end{aligned} \tag{3}$$

where, w and b are weight and bias respectively. Anything above the support vector is labeled as 1. Similarly, anything below as -1.

$$\begin{aligned} wx_i + b \geq +1 \quad for \quad y_i = +1 \\ wx_i + b \leq -1 \quad for \quad y_i = -1 \end{aligned} \tag{4}$$

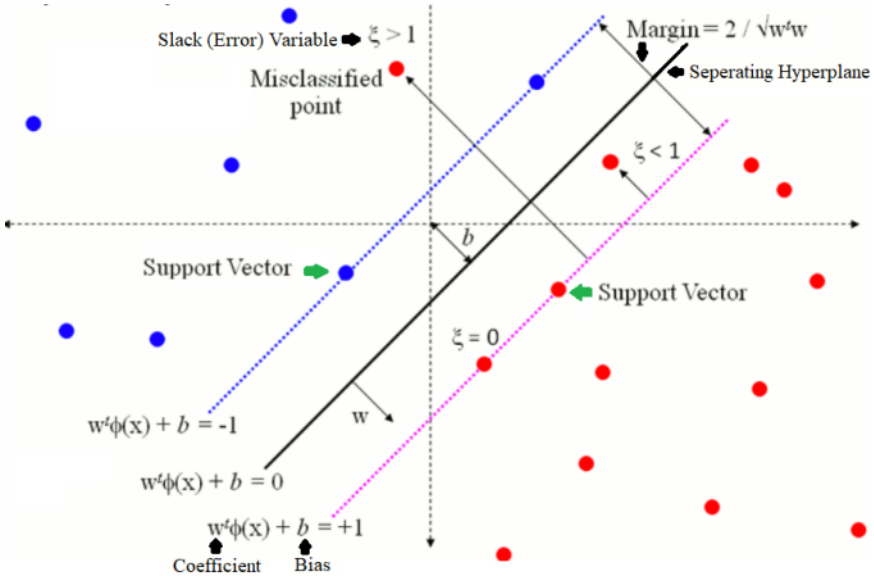

Figure 2: Support Vector Machines (SVM) Representation

We can combine both terms in equation (4) into one set of inequalities:

$$y_i(wx_i + b) - 1 \geq 0 \quad \forall i \tag{5}$$

The shortest distance between the two boundaries (margin) is $\frac{2}{||w||}$ (i.e., $\frac{2}{\sqrt{w^T w}}$). We want to maximize the margin, which is equivalent to minimizing $\frac{\sqrt{w^T w}}{2}$ (i.e., $\frac{w^T w}{2}$).

The problem of finding a maximum margin hypothesis for linearly separable data set is equivalent to the constrained minimization problem:

$$\begin{aligned} \min_{w,b} \quad & \frac{||w||^2}{2} \\ \text{s.t.} \quad & y_i(wx_i + b) \geq 1 \quad \forall i \end{aligned} \tag{6}$$

We construct the Langrangian with $\alpha_i \geq 0$ as multipliers to solve equation (6):

$$L_p \equiv \frac{||w||^2}{2} - \sum_{i=1}^{l} \alpha_i y_i (x_i w + b) + \sum_{i=1}^{l} \alpha_i \tag{7}$$

We minimize L_p in respect to w and b and simultaneously require that the derivatives

of L_p in respect to all the α_i, vanish, all subject to the constraint $\alpha_i \geq 0$. The involved derivation gives the following result:

$$\begin{aligned} w &= \sum_{i=1} \alpha_i y_i x_i \\ \sum_{i=1} \alpha_i y_i &= 0 \end{aligned} \tag{8}$$

The dual of the optimization problem equation (6) is:

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^m} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j x_i . x_j \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad \sum_{i=1}^{m} \alpha_i y_i = 0 \end{aligned} \tag{9}$$

The simple maximum margin framework, as outlined above, has two significant drawbacks. First, the decision boundary of the resulting classifier must be a linear combination of the features; and, second, the training set must be linearly separable. The SVM literature contains numerous solutions to these problems. First, even if the data is not linearly separable with the original features, it may become linearly separable if new non-linear features are constructed. Thus, instead of working directly with feature vector x as above, we apply a non-linear transformation Φ mapping x to $\Phi(x)$ in a higher dimensional space, and attempt to find a maximum margin separating plane in that space.

Note that the dual form equation (9) depends on the feature vectors only through a dot product. Therefore, we use a kernel function, $K(x_i, x_j) = \Phi(x_i).\Phi(x_j)$, to avoid the computational expense of working in the high dimensional space. Kernel functions operate directly on the original features and only implicitly represent the transformation to the high dimensional space. Some popular kernel functions are (Srivastava and Bhambhu [2010]):

Linear: $K(x_i, x_j) = x_i^T . x_j$
Degree-d polynomial: $K(x_i, x_j) = (\gamma x_i^T . x_j + c)^d$, $\gamma > 0$
RBF: $K(x_i, x_j) = exp(-\gamma ||x_i - x_j||^2$, $\gamma > 0$
Sigmoid: $K(x_i, x_j) = tanh(\gamma x_i^T . x_j + c)$

where, γ, c, and d are kernel parameters.

Second, even after transformation to a higher dimensional space, outliers may prevent the data from being linearly separable. In the soft margin technique, each example, (x_i, y_i), has a slack variable, $\xi_i \geq 0$, that measures how much the margin for that example falls below the desired unit distance. We modify the primal minimization problem equation (6) to minimize both the value of w and the sum of the slack variables in order to resolve the problem:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{||w||^2}{2} + \frac{C}{m}\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & y_i(wx_i + b) \geq 1 - \xi_i \quad \forall i \end{aligned} \tag{10}$$

where C trades off the relative importance of slack variable minimization and margin maximization.

4.2.2 Decision Trees Ensemble 'XGBoost'

Researchers use decision trees for a variety of different classification problems. Decision trees classify data by letting it traverse a tree full of dividing questions until it finally reaches a 'leaf' (end node) that predicts something final about the data. Trees are popular because they are easy to visualize, scale well, are good with anomalies such as missing data and outliers, and can handle both continuous and discrete data. A disadvantage of decision trees is their sensitivity to variance in data. One method to avoid this is to let more than onetree classify the data, a so-called "tree ensemble". This is the model that XGBoost is built upon (Blomqvist [2018]). Chen and Guestrin [2016] provide more detailed mathematical explanation of decision trees.

Figure 3 shows an example of a decision tree (image is taken from internet). The training occurs mathematically by evaluating a model, through a so-called objective function, Obj. In trees in general, the objective function has a term l that represents training loss. Training loss describes how correct the model is compared to the observed data. XGBoost also

implements a regularization term, Ω. Regularization controls overfitting.

XGBoost uses a tree ensemble, ideally of weak learners, and applies boosting, which is the process of iterating through a number of steps and adding a new function each step based on the error from the previous estimation. Normal gradient boosting uses the principle of gradient descent to find the minimum of the objective function, through adding a new function in each step—boosting. XGBoost, does all of this, while considering a slightly different take on the regularization term, but most importantly it considers second-order derivatives (which leads to faster computations).

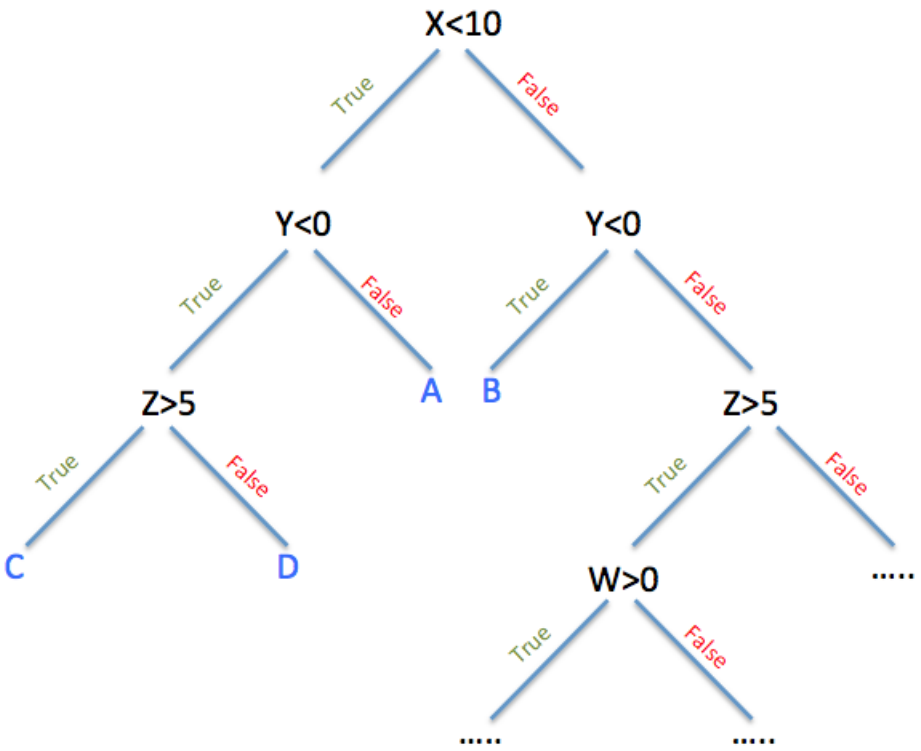

Figure 3: Decision Tree Representation

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k} \Omega(f_k), \quad f_k \in F \tag{11}$$

where the base models, f, can be found through boosting. The boosting starts with the

model formulating a prediction:

$$\hat{y_i}^{(0)} = 0 \tag{12}$$

XGBoost then adds functions (boosts it)

$$\hat{y_i}^{(1)} = f_1(x_i) = \hat{y_i}^{(0)} + f_1(x_i) \tag{13}$$

$$\hat{y_i}^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y_i}^{(1)} + f_2(x_i) \tag{14}$$

$$\hat{y_i}^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y_i}^{(t-1)} + f_t(x_i) \tag{15}$$

where, $\hat{y_i}^{(t)}$ is the model at step t. The next step is finding the base models, f. This is done through optimizing equation (11), which can be expressed as:

$$Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant \tag{16}$$

If we find the f_t that minimizes the expression, we find the f in each step in the boosting. We apply a Taylor expansion of equation (16) up to the second order and define the loss function L as the square loss. We can approximate equation (16) as:

$$Obj^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{17}$$

g_i and h_i are defined as the first- and second-order derivatives, respectively, of the loss function:

$$\begin{aligned} g_i &= \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \\ h_i &= \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \end{aligned} \tag{18}$$

We define the set of leaves, I_j, as

$$I_j = i | q(x_i) = j \tag{19}$$

We can expand the regularization to:

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{20}$$

where, T is the number of leaves in the tree; and, γ and λ are hyperparameters of the XGBoost model. The larger the respective values of λ and γ, the more conservative the algorithm becomes. We can write equation (17) as:

$$Obj^{(t)} = \sum_{j=1}^{T} [G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2] + \gamma T \tag{21}$$

where G_j and H_j are defined as

$$\begin{aligned} G_j &= \sum_{i \in I_j} g_i \\ H_j &= \sum_{i \in I_j} h_i \end{aligned} \tag{22}$$

The optimal leaf weight, as computed by minimizing equation (21) is:

$$w_j^* = \frac{-G_j}{H_j + \lambda} \tag{23}$$

The optimal loss value for the new model is:

$$Obj^* = \frac{-1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T \tag{24}$$

which is called the structure score - a measure of how good the tree structure is (the smaller the score, the better). It is not practically possible to find every feasible tree structure for a model. A greedy algorithm is the best way to grow; and, as the tree grows, it adds a split at each node. We calculate the score the tree gains to its objective function (i.e., a penalty) for a split as:

$$Gain = \frac{1}{2}[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} + \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}] - \gamma \tag{25}$$

where the first three terms are the "training loss reduction" and represent the score of the new left child, the right child, and the score if we do not split, respectively; and, the last term is a complexity penalty, or "regularization", from introducing a new leaf into the structure. If the total gain is negative (the added scores are smaller than γ), the tree structure loses performance by adding that leaf. If it is positive, the tree reduces its overall structure score and should perform the split, which is a variation of pruning the tree. An XGBoost model can also implement so-called recursive pruning, where the tree implements all possible splits to a maximum depth, and then recursively prunes the tree of the splits that created negative gain, so as to not overlook a split that looks like a loss at first but can lead to beneficial splits later.

5 Results and Discussion

There are 930 counties in the US Midwest; however, not all counties produce corn regularly. In fact, in our 1980 to 2018 study period, only 456 counties produced corn every year and 158 counties did not produce corn for more than five years. This suggests that weather in all of the counties does not matter in this analysis, and it indicates non-serious players only create model noise and their exclusion improves model accuracy.

We have a total of 18 weather variables recorded at a one-hour interval. We exclude a few variables that have high coefficient-of-correlation with other variables from further analysis. Soil variables are measured at county level but do not change over time. The embedded assumption is that soil characteristics do not vary much in the medium term. We select some soil variables using a correlation matrix result out of the 14 variables chosen based upon domain knowledge. Initially, we include soil variables in the training, but we later exclude them as they did not add any useful information to the model, probably due to time independency.

The objective of our article is to predict percentage change in daily CBOT corn futures prices (i.e., $[((Close_t/Close_{t-1}) - 1) * 100]$) or weekly futures prices (i.e., [((Close of the week/Open of the week) - 1)*100]). The scope of our study expands to forecasting the

price movement direction in the market. That is, we find if the daily CBOT corn futures price on day t is higher or lower than the price on day $t-1$ or if the closing price of a week is higher or lower than the opening price of that week. The opening price of a week is normally the Monday open price, but if the market is closed on a Monday then we use Tuesday's open price and likewise. The closing price of a week is generally the Friday close price, but if the market is not functioning that day then we use the close price from one day before.

5.1 Prediction Results

We apply a fully connected neural network as our base prediction model. However, our findings suggest that more specialized and focused training of the data by incorporating spatial attributes using a convolutional neural network (CNN) delivers better results. In a fully connected layer, each neuron connects to every neuron in the previous layer and each connection has its own weight. In contrast, in a convolutional layer, each neuron only connects to a few nearby neurons in the previous layer and every neuron uses the same set of weights. This ensures incorporation of the spatial features of the county wise data into the model.

A CNN framework requires input variables to be stored in a 3D format tensor cube. This is a very crucial step toward data preparation. The first axis of the 3D input structure represents the number of samples; the second axis is the number of steps, which is the number of US Midwest counties in our model; and, the third axis indicates input variables (weather variables in our case). The model structure with all 930 US Midwest counties provides a standard result, which is easily enhanced with changed format (i.e., a reduction in number of counties). We realize that a higher number of steps might be creating noise in our model; thus we try to reduce the number of steps with a different configuration. We collapse weather data at the state level, which brings the number of steps down to 10 (the number of states in our sample). However, our results do not improve much, probably due to over-averaging of the input data, which makes the data lose its spatial characteristics.

5.1.1 Percentage Change in Daily CBOT Corn Futures Prices

Figure 4 shows the Pearson correlation coefficient of daily weather variables. We exclude variables for which the absolute value of mutual correlation coefficient is greater than or equal to 0.7. We employ the remaining 11 variables along with PDSI and GDD to predict percentage change in daily CBOT corn futures prices.

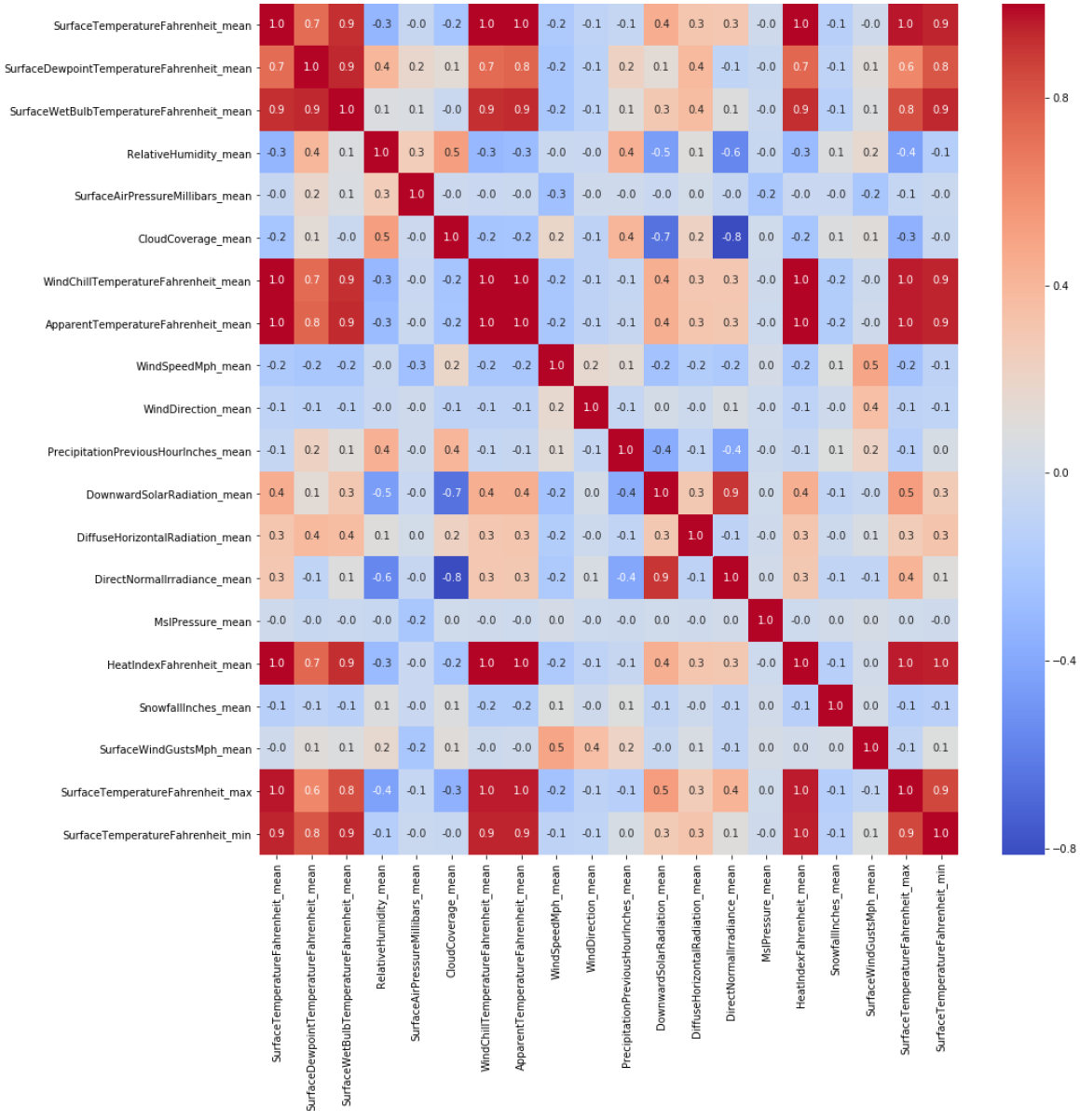

Figure 4: Coefficient of correlation of daily weather variables

Table 1 shows the model structure details along with optimized hyperparameters and accuracy level (i.e., mean square error (MSE) and normalized root mean square error (NRMSE)). We employ NRMSE to compare efficacy of two models when the models do not have identical test data sets. We define NRMSE as $\frac{RMSE}{max(test-data)-min(test-data)}$. We use these hyperparameters to evaluate our model, excluding counties that have not produced corn for more than five years, which leaves 772 counties in our model. In an attempt to improve our results further, we select only counties that produced corn every year of our study period (1980–2018), which leaves 456 counties in our model and improves the outcome.

We select the hyperparameters that produce the best results for our model (see table 1). The first axis of the 3D input variable, X_t, of the training set reflects the number of samples for training the model. The second and third axes are the number of counties and weather variables, respectively. The first axis of the 2D output variable, y_t, indicates sample size and the second axis shows output value, which is the percentage change in daily CBOT corn futures prices in this case. The test set axes have a similar definition. The normalized error for our model is 0.1264 (see table 1). The out-of-sample prediction results from 2012 to 2018 are very close to the results shown in table 1, which is for a random 20% split.

Figure 5 shows the predicted and actual (test set) value. The plot indicates that the model learns the normal or modest fluctuations but does not predict the extreme price changes well. We expect that incorporating temporal impact (i.e., adding history) in the input variable by stacking a few previous days could improve model learning capacity. We intend to use unanticipated weather variables, also known as weather surprises, instead of base variables for greater responsiveness of our model. We define a weather surprise as the difference between the current weather variable and the preceding few years' average of that variable for that month.

It is widely believed that not only does the current day's weather impact the market but historical weather could affect prices as well. The weather forecasting models in the market have gained credibility for predicting weather in the short run with good accuracy,

Table 1: Model Structure for Predicting Daily CBOT Corn Futures Price Changes

Variables	Hyperparameters	Results
Daily mean of: Surface Temperature, Relative Humidity, Cloud Coverage, Wind Speed, MslPressure, Snowfall, Precipitation PreviousHour, Wind Direction, Surface Air Pressure, Horizontal Radiation Diffusion, Surface Wind Gusts, GDD, and PDSI.	4 Conv1D layers with BatchNormalization and MaxPooling (size = 2): kernel size = 5, 3, 3, 3; Padding = same for all; Activation = linear for all. 2 Dense layers: Neurons = 64, 1; Activation = linear for all. Optimizer = Adam. Epochs = 1000	MSE: 2.58; NRMSE: 0.1264
Test size = 0.20; Number of steps (counties) = 456		Train set shape: (4526, 456, 13) (4526, 1); Test set shape: (1132, 456, 13) (1132, 1)

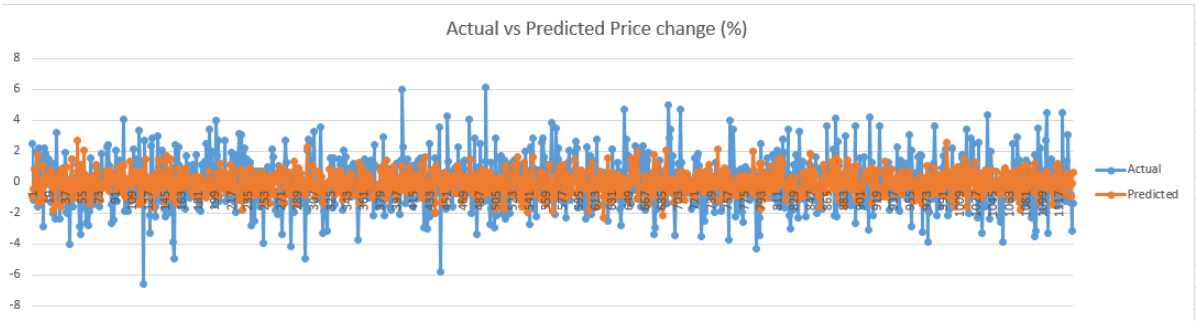

Figure 5: Percentage change in daily CBOT corn futures prices (%) using model described in table 1

which gives traders access to very credible expected future weather information as well. Imperatively, we include few historical and future days in our model structure. We define three model structures - one has only historical weather information, one considers only future days, and one incorporates both historical and future weather data. Notably, all three models include current day weather along with other variables as previously described. Table 2 shows the three model structures outlined here with corresponding results. We only display NRMSE and not MSE, as the test sets are not the same, a consequence owing to

different model structures. The input variables and model hyperparameters are the same as described in table 1.

Table 2: Model Structures Incorporating Temporal Impact

Structures	Data Input	Results
1 (Backward looking)	Using X_{t-9} to X_t to predict y_t	NRMSE: 0.1285
2 (Forward looking)	Using X_t to X_{t+9} to predict y_t	NRMSE: 0.1244
3 (Backward and forward looking)	Using X_{t-4} to X_{t+5} to predict y_t	NRMSE: 0.1187

Figure 6 shows the predicted and actual (test set) value for model structure 3, as explained in table 2, which exhibits a relatively better result. This indicates that traders in the market use past as well as future weather information. It is also evident from table 2 that traders are more responsive to the future weather than past weather.

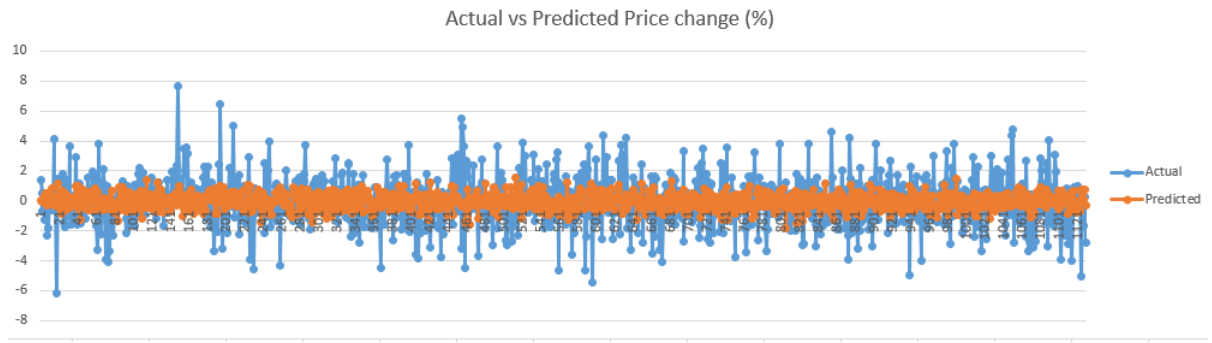

Figure 6: Percentage change in daily CBOT corn futures prices (%) using model 3 of table 2

5.1.2 Percentage Change in Weekly CBOT Corn Futures Prices

Table 3 displays our model structure details, optimized hyperparameters, and accuracy level (i.e., the MSE and NRMSE). Our model contains 10 input variables and has a total of 1,926 samples. We use weather data for all months of a year against previous models that only employ data for growing season months. There are two reasons for this change.

First, using weekly price changes instead of daily significantly reduces sample size. If we keep data only for April to October, we lose approximately one-third of the total data points. Second, weather outside of the growing season can have some indirect impact on a crop's market availability by affecting attributes like soil health and grain storage and transportation.

Table 3: Model Structure for Predicting Weekly CBOT Corn Futures Price Changes

Variables	Hyperparameters	Results
Weekly mean of the daily mean of: Surface Temperature, Relative Humidity, Cloud Coverage, Wind Speed, MslPressure, Snowfall, Precipitation PreviousHour, Wind Direction; Weekly max of the daily mean of Relative Humidity; Weekly max of the daily max of Surface Temperature.	4 Conv1D layers with BatchNormalization and MaxPooling (size = 2): kernel size = 5, 3, 3, 3; Padding = same for all; Activation = linear for all. 2 Dense layers: Neurons = 64, 1; Activation = linear for all. Optimizer = Adam. Epochs = 1000	MSE: 9.4; NRMSE: 0.1189; Correlation coefficient between actual and predicted value: 0.2
Test size = 0.20; Number of steps (counties) = 456		Train set shape: (1540, 456, 10) (1540, 1); Test set shape: (386, 456, 10) (386, 1)

Figure 7 shows the predicted and actual (test set) value and indicates that the model learns the usual fluctuations, which the significant value of correlation coefficient between actual and predicted data also confirms. However, the predicted value does not capture the extreme price movements (larger spikes in actual data). We plan to add some more historical information into the input variable by including weather variables of previous week as well to increase efficacy of the model. Also, we intend to use weather variables of previous week and next week for predicting current week price changes to investigate the impact of past and future weather information respectively.

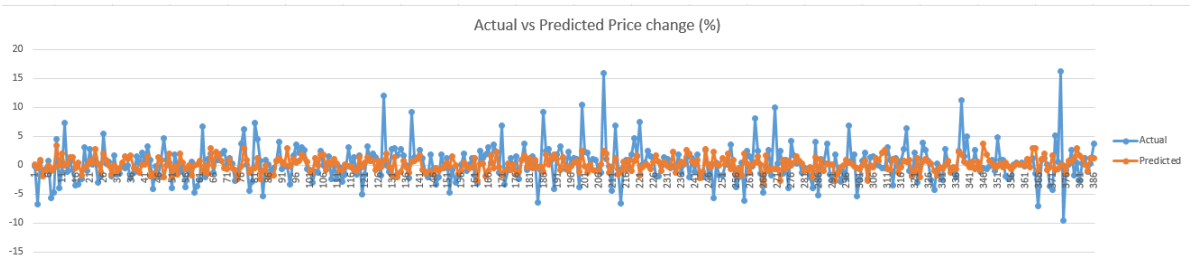

Figure 7: Percentage change in weekly CBOT corn futures prices (%) using model shown in table 3

5.2 Classification Results

Table 4 shows the accuracy results of the SVM and XGBoost models and the structures involved. The variables column of table 3 details the 10 weather variables we use in our models. Test size is 0.2, as before. We employ these models to find the direction of price change. Decision tree ensemble model XGBoost demonstrates quite a promising outcome - it correctly predicts the direction of the price movement in more than 60% of test cases. Notably, the data set is balanced. The XGBoost out-of-sample classification results for 2012–2018 are very close to the results shown in table 4, which is for random 20% split.

Table 4: Classification Results for Percentage Change in Weekly Prices

Models	Hyperparameters	Data Shape	Accuarcy (%)
Support Vector Machine	kernel = 'sigmoid', gamma = 'auto'	Train set shape: (1540, 456*10) (1540, 1); Test set shape: (386, 456*10) (386, 1)	56
XGBoost	booster = 'gblinear', no. of estimators = 1000	Train set shape: (1540, 456*10) (1540, 1); Test set shape: (386, 456*10) (386, 1)	60

6 Conclusion

Our study informs the traders about the relevance of weather information. It evaluates which time frame of weather information is important for the market. Prediction models

that forecast percentage change in prices and classification models that predict the direction of price changes are very useful to traders in the futures market. Traders can utilize these statistics to outperform the market and make profits. Traders can use information about price movement direction to decide about going long or short in the market. We intend to try different model structures and hybrid algorithms to enhance the accuracy of our models.

References

I. Almeida, M. Hirtzer, and L. Mulvany. Corn futures post worst rout since 2013 on u.s. acreage surprise, August 2019.

J. Blomqvist. Using xgboost to classify thebeihang keystroke dynamics database, 2018. ISSN 1401-5757.

C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

E. Ceperic, S. Žiković, and V. Ceperic. Short-term forecasting of natural gas prices using machine learning and feature selection algorithms. *Energy*, 140, 09 2017. doi: 10.1016/j.energy.2017.09.026.

T. Chen and C. Guestrin. Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2016. doi: 10.1145/2939672.2939785. URL `http://dx.doi.org/10.1145/2939672.2939785`.

A. V. Dam, L. Karklis, and T. Meko. After a biblical spring, this is the week that could break the corn belt, June 2019.

S. Goutham, S. Kp, and V. R. Automated detection of diabetes using cnn and cnn-lstm network and heart rate signals. *Procedia Computer Science*, 132:1253–1262, 01 2018. doi: 10.1016/j.procs.2018.05.041.

Z. Jiang, C. Liu, N. P. Hendricks, B. Ganapathysubramanian, D. J. Hayes, and S. Sarkar. Predicting county level corn yields using deep long short term memory models, 2018.

N. Kantanantha, N. Serban, and P. Griffin. Yield and price forecasting for stochastic crop decision planning. *Journal of Agricultural, Biological, and Environmental Statistics*, 15: 362–380, 09 2010. doi: 10.1007/s13253-010-0025-7.

S. Khaki, L. Wang, and S. Archontoulis. A cnn-rnn framework for crop yield prediction, 11 2019.

T. Kim and H. Kim. Forecasting stock prices with a feature fusion lstm-cnn model using different representations of the same data. *PLoS One*, 14(2), Feb. 2019. ISSN 1932-6203. doi: 10.1371/journal.pone.0212320.

W. Kristjanpoller and M. Minutolo. Gold price volatility: A forecasting approach using the artificial neural network–garch model. *Expert Systems with Applications*, 42, 05 2015. doi: 10.1016/j.eswa.2015.04.058.

S. Kulkarni and I. Haidar. Forecasting model for crude oil price using artificial neural networks and commodity futures prices. *arXiv.org, Quantitative Finance Papers*, 2, 06 2009.

P. Limpaphayom, P. R. Locke, and P. Sarajoti. Gone with the wind: Chicago's weather and futures trading. *Review of Futures Markets*, 2007.

S. Lodha, E. Kreps, D. Helmbold, and D. Fitzpatrick. Aerial lidar data classification using support vector machines (svm). pages 567–574, 06 2006. doi: 10.1109/3DPVT.2006.23.

W. Long, Z. Lu, and L. Cui. Deep learning-based feature engineering for stock price movement prediction. *Knowledge-Based Systems*, 164, 11 2018. doi: 10.1016/j.knosys.2018.10.034.

J. Lu and R. K. Chou. Does the weather have impacts on returns and trading activities in order-driven stock markets? evidence from china. *Journal of Empirical Finance*, 19(1): 79–93, 2012.

B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001. ISBN 0262194759.

D. Srivastava and L. Bhambhu. Data classification using support vector machine. *Journal of Theoretical and Applied Information Technology*, 12:1–7, 02 2010.

M. Teow. Understanding convolutional neural networks using a minimal model for handwritten digit recognition. pages 167–172, 10 2017. doi: 10.1109/I2CACIS.2017.8239052.

R. Yamashita, M. Nishio, R. Do, and K. Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9, 06 2018. doi: 10.1007/s13244-018-0639-9.